ʕ •ᴥ• ʔ

b34r5hell

PWN

Agenda

> What is PWN
> Prerequisite Knowledge
> Stack
> Buffer Overflow
> Other Techniques

ʕ•ᴥ•ʔ

What is PWN

- PWN - utterly defeat (an opponent or rival); completely get the better of.

- **Important Aspects**
  - Binary exploitation (focus of today)
  - Web exploitation

ʕ•ᴥ•ʔ

How does it apply to CTFs?

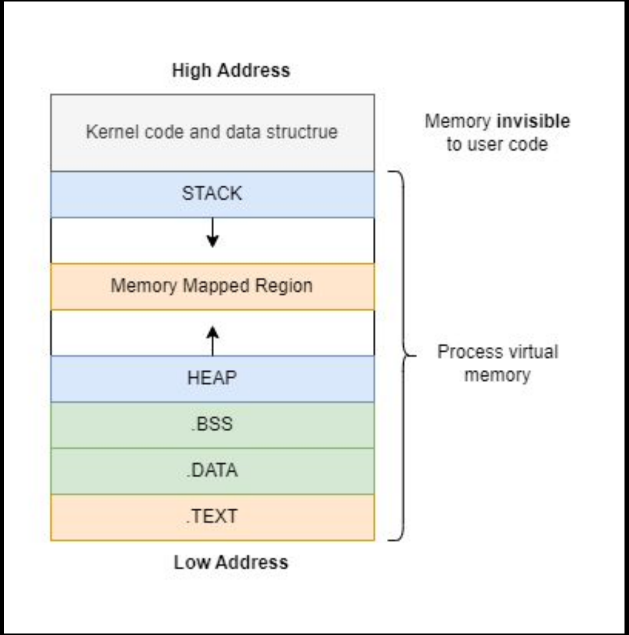
- **Have to exploit some vulnerability on a server run by the CTF**
    - Use the exploit to run arbitrary code on the server and access the flag

- Will usually also be given a local copy of the executable along with any relevant libraries
    - Can develop exploit locally (e.g. buffer overflow, return-oriented-programming, format string) and use debugging tools
    - When it works locally, can send exploit data to server

ʕ•ᴥ•ʔ

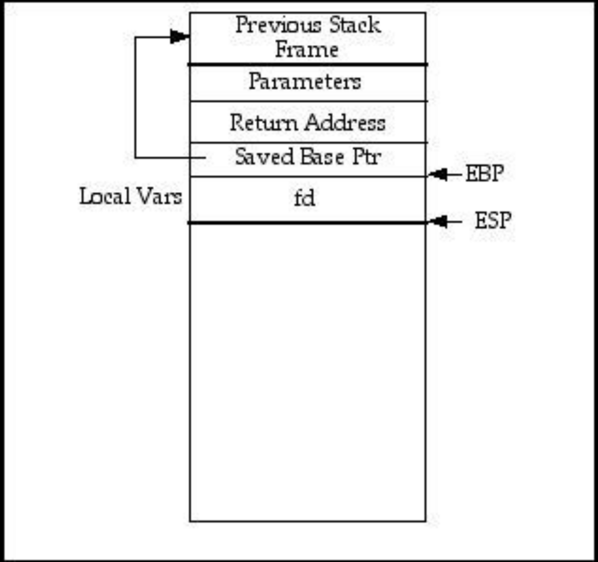
Prerequisite Knowledge

- Systems knowledge (CSE361 is a great class to learn this)
  - Basic Assembly (x86-64)
  - Program memory mapping and data structures
    - Stack
    - Heap
    - Dynamic linking (PLT table)
  - C language
- Reverse Engineering
  - Have to find the vulnerabilities



Stack

- Named stack because it is a stack data structure
- Region of memory used to store local function data
- Each function has its own frame, where it stores local variables used in that function
- Low address at the bottom:



ʕ•ᴥ•ʔ

Buffer Overflow

- **A program allocates x amount of bytes for a buffer**
  - A buffer is just a specific portion of memory you will later fill up with data, usually from user-input

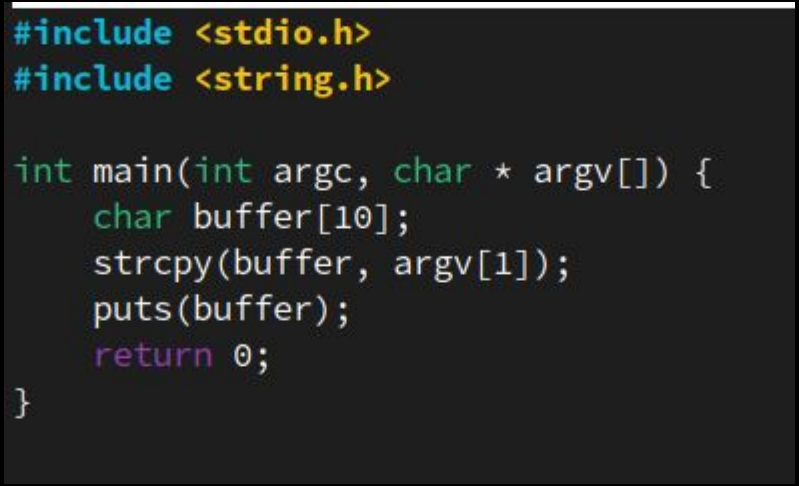
- **In an insecure implementation, the buffer can be filled with more than x bytes**
  - Bytes are written into other parts of memory not allocated to the intended buffer

- **The basis for many common types of attacks/techniques**
  - Stack overflow
  - Heap overflow
  - Return-oriented programming
  - etc.

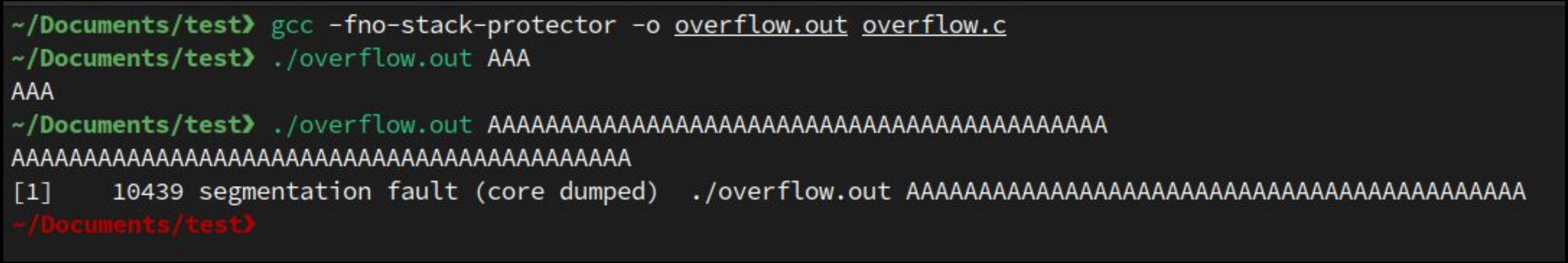
ʕ•ᴥ•ʔ

Simple C Example

- Certain (old) C functions don’t have any inherent protections
    - strcpy()
    - gets()
    - etc.



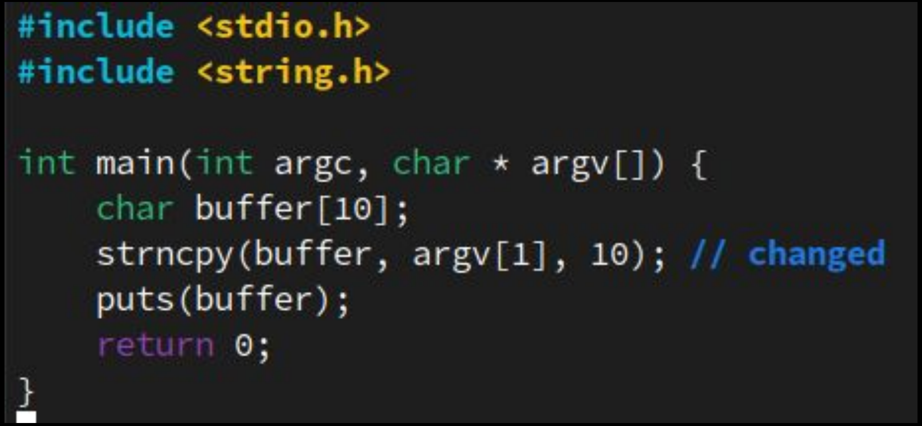
```c
#include <stdio.h>
#include <string.h>

int main(int argc, char * argv[]) {
    char buffer[10];
    strcpy(buffer, argv[1]);
    puts(buffer);
    return 0;
}
```



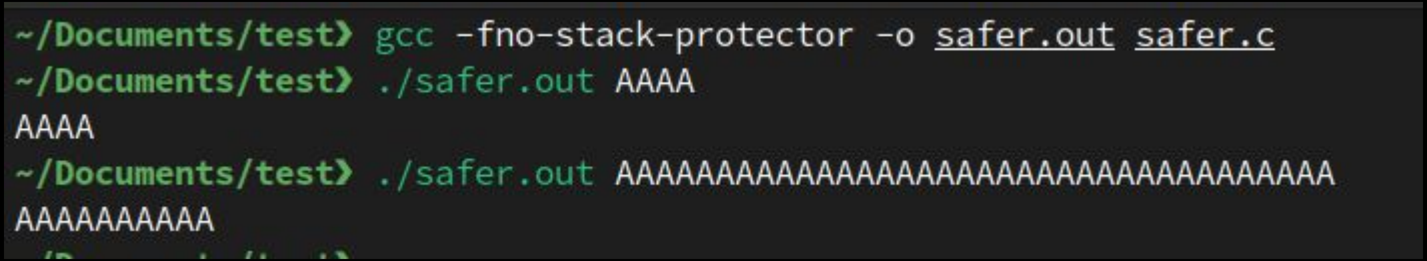
```
~/Documents/test❯ gcc -fno-stack-protector -o overflow.out overflow.c
~/Documents/test❯ ./overflow.out AAA
AAA
~/Documents/test❯ ./overflow.out AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[1]    10439 segmentation fault (core dumped)  ./overflow.out AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
~/Documents/test❯
```

Making it Safe

- Other functions have *some* inherent protections by cutting off the input
  - strncpy()
  - fgets()
  - etc.



```c
#include <stdio.h>
#include <string.h>

int main(int argc, char * argv[]) {
    char buffer[10];
    strncpy(buffer, argv[1], 10); // changed
    puts(buffer);
    return 0;
}
```



```
~/Documents/test❯ gcc -fno-stack-protector -o safer.out safer.c
~/Documents/test❯ ./safer.out AAAA
AAAA
~/Documents/test❯ ./safer.out AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAA
```

Other Protections

- What if you can’t trust developers to make secure programs?
  - Solution: Built-in system protections

- Non-executable Stack
  - Prevents an attacker from inserting their own code
- Address space layout randomization (ASLR) and Position-independent executable (PIE)
  - Limits the resources available to create an exploit
- Stack Canary
  - Detects buffer overflows before they can take control
  - Can be brute forced, byte-by-byte

- Even with all of these, you can still sometimes exploit a program, it's just harder and requires the program to have multiple or particularly bad mistakes

ʕ•ᴥ•ʔ

Other Techniques and Vulnerabilities

- Vulnerabilities
  - Format String vulnerability
  - Race Conditions
  - Heap Overflow (and other Heap vulnerabilities)
- Techniques
  - Return Oriented Programming
  - GOT-PLT Redirection
  - Canary Brute forcing (byte-by-byte)

ʕ•ᴥ•ʔ

Tasks

- Complete the module in the dojo

ʕ•ᴥ•ʔ